**MOBILE AUTOMATION**

**CUCUMBER – RUBY - CALABASH**

**Overview**

QA team performed a research about mobile automation tools, found in this research two tools for mobile automation: Appium and Calabash. After a spike with Appium and Calabash, we decidade to use Calabash since it is simpler to integrate Cucumber with Ruby and Calabash, than integrate Cucumber with Java and Appium for the automation project. Also has been noticed that using Calabash the performance of the iOS simulator is better at the moment of running the automated test.

**Process**

For automation testing, we will select at first the acceptance test cases of the application; this means the more important test cases. We will implement these test cases in plain text using Gherkin language in a feature file (.feature), and then we will implement the definitions of these steps in ruby language using calabash libraries to interact with our iOS application. Once this is done, we will run these acceptance tests for each new released build, once a new feature is implemented, this in order to catch the bugs faster and also to check if any functionality already implemented is broken with the new changes.

**Base automation framework**

1. **Prepare environment.**

Reference:

<https://www.moncefbelyamani.com/ios-automated-testing-with-calabash-cucumber-ruby/>

**1.1. Install RVM**

curl -L https://get.rvm.io | bash -s stable --ruby

**1.2. To Start using RVM:**

source /Users/username/.rvm/scripts/rvm

**1.2. Install calabash-cucumber gem**

gem install calabash-cucumber

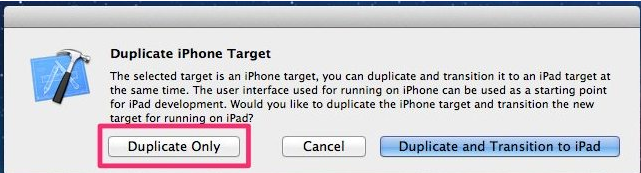
**1.3. Generate the features folder and irb shell scripts**

$ calabash-ios gen

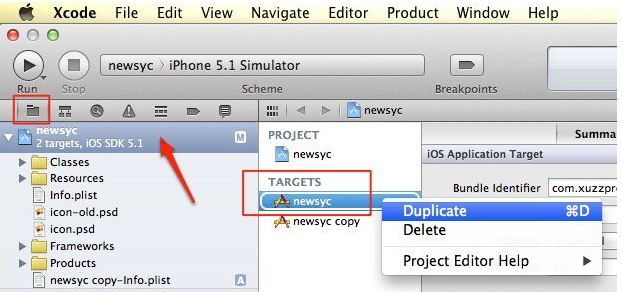
This command will generate the skeleton of the calabash-cucumber project.

**1.4. Set up your Xcode project manually to Set up Calabash Server**

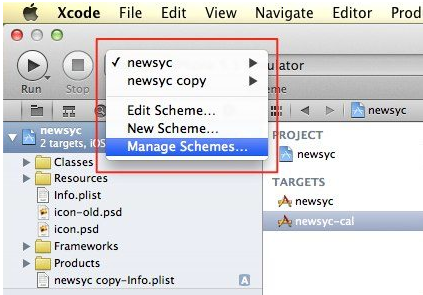
* Open Xcode
* Select File->Open (⌘O), and open your project folder.
* Click on your project in the leftmost pane, then right-click (or two-finger tap) on your project under TARGETS, and select Duplicate. Or, simply click on your project and press ⌘D. If you get a Duplicate iPhone Target prompt, click on the Duplicate Only button.



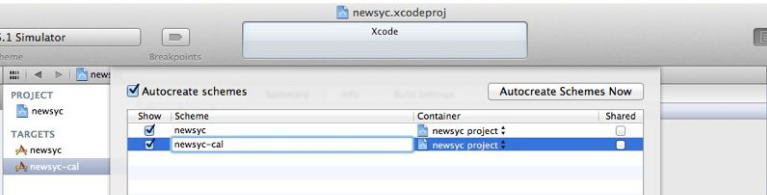
This should result in a new target called *yourproject-copy*, as shown in the screenshot below:



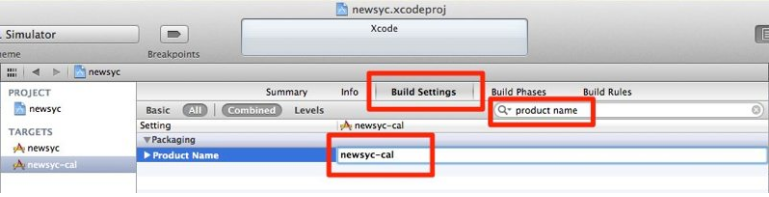
* Double-click on *yourproject-copy* to put it in edit mode, and rename it to *yourproject-calabash*, or *yourproject*-*cal* for short. The name you choose doesn’t really matter, as long as it makes sense to you.
* Click on your*project* in the dropdown at the top left, to the right of the “Stop” button, and select *Manage Schemes…*:



* Click once on *yourproject copy*, then wait, then click once again, making sure to click on the name itself, not anywhere else on that row. This will put you in edit mode, and you can rename the scheme to *yourproject-cal*.



* Click *OK*, then click on *Build Settings* in the center pane, search for “product name”, then double-click on *yourproject copy* and rename it to *yourproject-cal*:

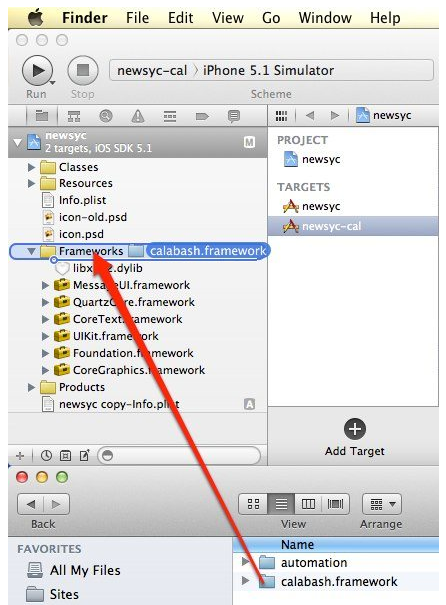


* Download the latest version of calabash-ios from <https://github.com/calabash/calabash-ios/downloads>.

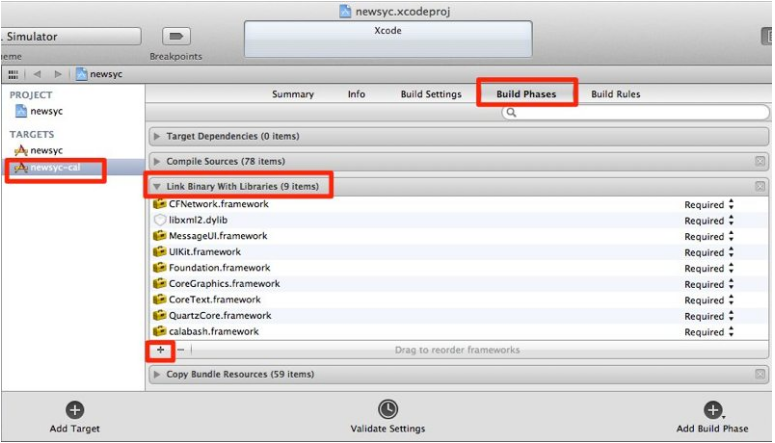
If the link does not work, you can also download it directly, go to your Xcode Project folder and from command line use:

*calabash-ios download*

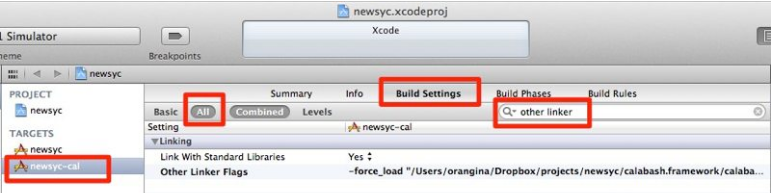
Unzip the file (you will end up with a folder called *calabash.framework*), then position the Finder window in one half of the screen, and the Xcode window in the other half. Drag and drop the *calabash.framework* folder into Frameworks folder.



* Once you drop the *calabash.framework* folder in Xcode, you will be prompted to “choose options for adding these files”. Make sure the following are true: *Copy items into destination group's folder (if needed)* is checked, *Create groups for any added folders* is selected, and only your newly-created target (in our case newsyc-cal) is checked.
* Click *Finish*, then click on the *yourproject-cal* target, click on *Build Phases*, expand *Link Binary with Libraries*, click on *+*, click on *CFNetwork.framework*, and click on *Add*. You should end up with something very similar to this:



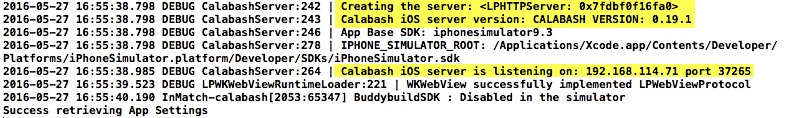
With the *yourproject-cal* target still selected, click on *Build Settings*, click on *All* (if it’s not already selected), then search for other linker, click on the *Other Linker Flags* row, then click once under Yes to enable edit mode, and copy and paste the following: *-force\_load "$(SRCROOT)/calabash.framework/calabash" -lstdc++*. Click anywhere outside of the text field to save your changes. You should end up with something like this (note that what appears after *-force\_load* will be different for you since that is the path to your project on your computer):



**1.5. Test your setup.**

In the Scheme dropdown in the top left (to the right of the “Stop” button), select the *yourproject-cal* target on the left side of the dropdown, select *iPhone 6 Plus Simulator* on the right side of the dropdown, then click the *Run* button (or press ⌘R).

While the iOS Simulator is launching, click on the middle button in the *View* section in the top right of Xcode to display the console output at the bottom of the center pane. Once the app is ready to launch in the Simulator, you will receive a dialog asking if you want the application to accept incoming network connections. Click *Allow*, and when you see something like this in the console output, you’re good to go:



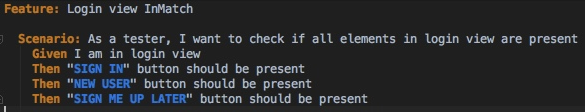
**2. Write test cases in Gherkin language - plain text.**

For this project, we will write the automated test cases in plain text using cucumber.

Cucumber executes your *.feature* files, and those files contain executable specifications written in a language called Gherkin.

Gherkin is plain-text English (or one of 60+ other languages) with a little extra structure. Gherkin is designed to be easy to learn by non-programmers, yet structured enough to allow concise description of examples to illustrate business rules in most real-world domains.

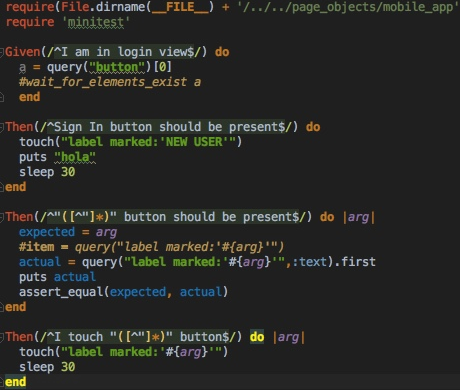
Below an example of an automated test case written in Gherkin:



**3. Write the steps definitions for the Gherkin steps**

After writing the test cases in plain text, we need to write the step definitions to interact with the elements of our iOS/Android application (also in this steps we can communicate with database or backend if needed).

For this we write the steps definitions in a rb file as shown in the InMatch example below:



**4. Run the automated test cases**

To execute our automated test cases, we just need to write the command *cucumber* in the command line when we are located in our automation project folder:

